UNIT – V

Reinforcement Learning

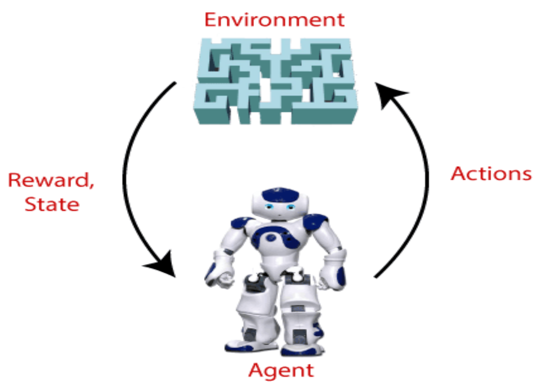
Syllabus:

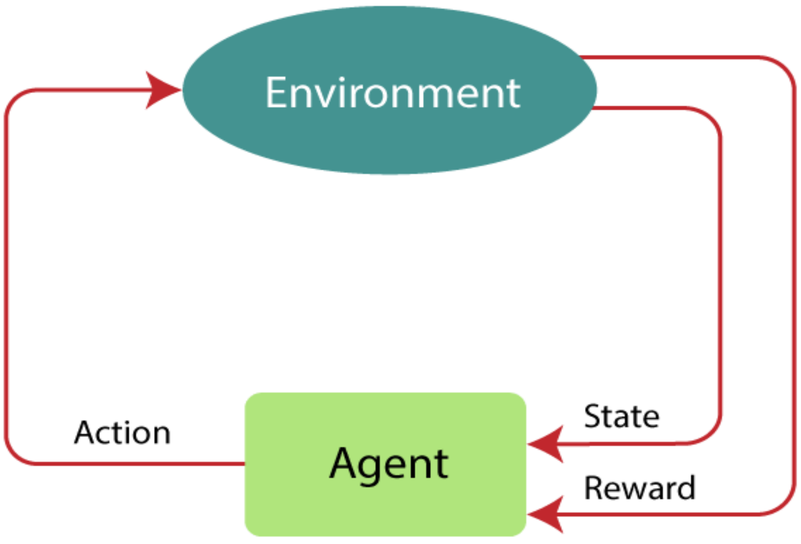
- Reinforcement Learning:
- Passive reinforcement learning
    - Direct Utility Estimation,
    - Adaptive Dynamic Programming (ADP),
    - Temporal Difference Learning,
- Active reinforcement learning
    - Q learning.

Introduction:

- Reinforcement Learning is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions. For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty. It is about taking **suitable action to maximize reward** in a particular situation.
- In Reinforcement Learning, the agent learns automatically using feedbacks without any labeled data, unlike supervised learning.
- Since there is no labeled data, so the agent is bound to learn by its experience only.
- RL solves a specific type of problem where decision making is sequential, and the goal is long-term, such as game-playing, robotics, etc.
- The agent interacts with the environment and explores it by itself.
- The primary goal of an agent in reinforcement learning is to improve the performance by getting the maximum positive rewards.
- The agent learns with the process of hit and trial, and based on the experience, it learns to perform the task in a better way. Hence, we can say that *"Reinforcement learning is a type of machine learning method where an intelligent agent (computer program) interacts with the environment and learns to act within that."* How a Robotic dog learns the movement of his arms is an example of Reinforcement learning.
- It is a core part of Artificial intelligence, and all AI agent works on the concept of reinforcement learning. Here we do not need to pre-program the agent, as it learns from its own experience without any human intervention.
- Example: Suppose there is an AI agent present within a maze environment, and his goal is to find the diamond. The agent interacts with the environment by performing some actions, and based on those actions, the state of the agent gets changed, and it also receives a reward or penalty as feedback.
- The agent continues doing these three things (take action, change state/remain in the same state, and get feedback), and by doing these actions, he learns and explores the environment.

- The agent learns that what actions lead to positive feedback or rewards and what actions lead to negative feedback penalty. As a positive reward, the agent gets a positive point, and as a penalty, it gets a negative point.





Terms used in Reinforcement Learning

- **Agent:** An entity that can perceive/explore the environment and act upon it.
- **Environment:** A situation in which an agent is present or surrounded by. In RL, we assume the stochastic environment, which means it is random in nature.
- **Action:** Actions are the moves taken by an agent within the environment.
- **State:** State is a situation returned by the environment after each action taken by the agent.
- **Reward:** A feedback returned to the agent from the environment to evaluate the action of the agent.
- **Policy:** Policy is a strategy applied by the agent for the next action based on the current state.
- **Value:** It is expected long-term retuned with the discount factor and opposite to the short-term reward.
- **Q-value:** It is mostly similar to the value, but it takes one additional parameter as a current action (a).

Key Features of Reinforcement Learning

- In RL, the agent is not instructed about the environment and what actions need to be taken.
- It is based on the hit and trial process.
- The agent takes the next action and changes states according to the feedback of the previous action.
- The agent may get a delayed reward.
- The environment is stochastic, and the agent needs to explore it to reach to get the maximum positive rewards.

Approaches to implement Reinforcement Learning

There are mainly three ways to implement reinforcement-learning in ML, which are:

1. **Value-based:**
   The value-based approach is about to find the optimal value function, which is the maximum value at a state under any policy. Therefore, the agent expects the long-term return at any state(s) under policy π.

2. **Policy-based:**
   Policy-based approach is to find the optimal policy for the maximum future rewards without using the value function. In this approach, the agent tries to apply such a policy that the action performed in each step helps to maximize the future reward.

   The policy-based approach has mainly two types of policy:
   Deterministic: The same action is produced by the policy (π) at any state.
   Stochastic: In this policy, probability determines the produced action.

3. **Model-based:** In the model-based approach, a virtual model is created for the environment, and the agent explores that environment to learn it. There is no particular solution or algorithm for this approach because the model representation is different for each environment.

Types of Reinforcement learning

There are mainly two types of reinforcement learning, which are:

- Positive Reinforcement
- Negative Reinforcement

Positive Reinforcement

- Positive reinforcement learning is the process of encouraging or adding something when an expected behavior pattern is exhibited to increase the likelihood of the same behavior being repeated. (or)
- The positive reinforcement learning means adding something to increase the tendency that expected behavior would occur again. It impacts positively on the behavior of the agent and increases the strength of the behavior.
- It is a recurrence of behaviour due to positive rewards.
- Rewards increase strength and the frequency of a specific behaviour.
- This encourages to execute similar actions that yield maximum reward.

- This type of reinforcement can sustain the changes for a long time, but too much positive reinforcement may lead to an overload of states that can reduce the consequences.
- For example, if a child passes a test with impressive grades, they can be positively reinforced with an ice cream cone.

Negative Reinforcement

- Negative reinforcement involves increasing the chances of specific behavior to occur again by removing the negative condition.
- For example, if a child fails a test, they can be negatively reinforced by taking away their video games. This is not precisely punishing the child for failing, but removing a negative condition (in this case, video games) that might have caused the kid to fail the test.
- The negative reinforcement learning is opposite to the positive reinforcement as it increases the tendency that the specific behavior will occur again by avoiding the negative condition.
- It can be more effective than the positive reinforcement depending on situation and behavior, but it provides reinforcement only to meet minimum behavior.

Elements of Reinforcement Learning

There are four main elements of Reinforcement Learning, which are given below:

Policy
Reward Signal
Value Function
Model of the environment

1) Policy: A policy can be defined as a way how an agent behaves at a given time. It maps the perceived states of the environment to the actions taken on those states. A policy is the core element of the RL as it alone can define the behavior of the agent. In some cases, it may be a simple function or a lookup table, whereas, for other cases, it may involve general computation as a search process. It could be deterministic or a stochastic policy:

For deterministic policy: $a = \pi(s)$.

For stochastic policy: $\pi(a \mid s) = P[At =a \mid St = s]$

2) Reward Signal: The goal of reinforcement learning is defined by the reward signal. At each state, the environment sends an immediate signal to the learning agent, and this signal is known as a reward signal. These rewards are given according to the good and bad actions taken by the agent. The agent's main objective is to maximize the total number of rewards for good actions. The reward signal can change the policy, such as if an action selected by the agent leads to low reward, then the policy may change to select other actions in the future.

3) Value Function: The value function gives information about how good the situation and action are and how much reward an agent can expect. A reward indicates the immediate signal for each good and bad action, whereas a value function specifies the good state and action for the future. The value function depends on the reward as, without reward, there could be no value. The goal of estimating values is to achieve more rewards.

4) Model: The last element of reinforcement learning is the model, which mimics the behavior of the environment. With the help of the model, one can make inferences about how the environment will behave. Such as, if a state and an action are given, then a model can predict the next state and reward.

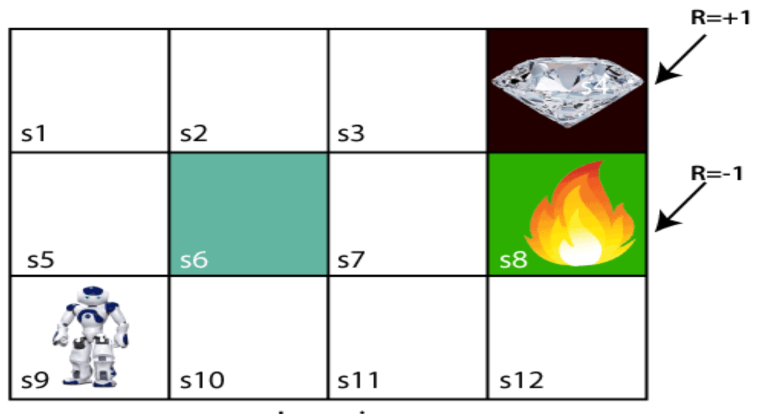
The model is used for planning, which means it provides a way to take a course of action by considering all future situations before actually experiencing those situations. The approaches for solving the RL problems with the help of the model are termed as the **model-based approach**. Comparatively, an approach without using a model is called a **model-free approach**.

How does Reinforcement Learning Work?

To understand the working process of the RL, we need to consider two main things:

- Environment: It can be anything such as a room, maze, football ground, etc.
- Agent: An intelligent agent such as AI robot.

Let's take an example of a maze environment that the agent needs to explore. Consider the below image:



Lorem ipsum

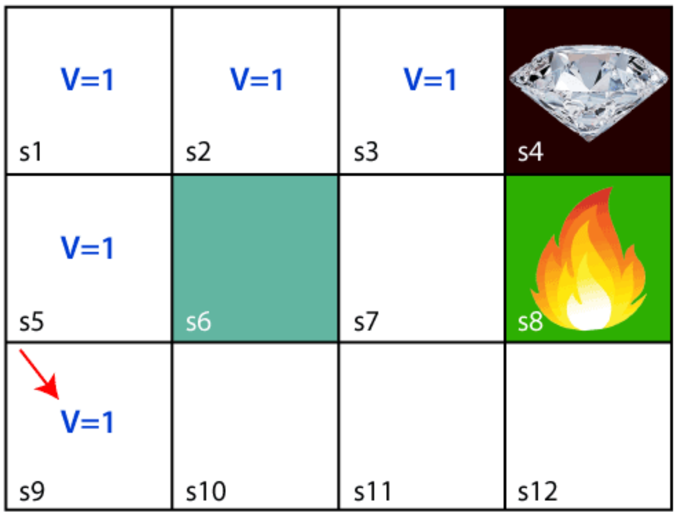
In the above image, the agent is at the very first block of the maze. The maze is consisting of an S6 block, which is a wall, S8 a fire pit, and S4 a diamond block.

The agent cannot cross the S6 block, as it is a solid wall. If the agent reaches the S4 block, then get the +1 reward; if it reaches the fire pit, then gets -1 reward point. It can take four actions: move up, move down, move left, and move right.

The agent can take any path to reach to the final point, but he needs to make it in possible fewer steps. Suppose the agent considers the path S9-S5-S1-S2-S3, so he will get the +1-reward point.

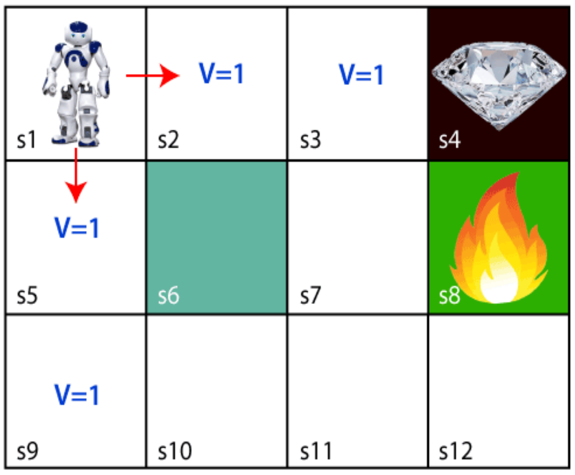
The agent will try to remember the preceding steps that it has taken to reach the final step. To memorize the steps, it assigns 1 value to each previous step.

Consider the below step:



Now, the agent has successfully stored the previous steps assigning the 1 value to each previous block. But what will the agent do if he starts moving from the block, which has 1 value block on both sides?

Consider the below diagram:



It will be a difficult condition for the agent whether he should go up or down as each block has the same value. So, the above approach is not suitable for the agent to reach the destination. Hence to solve the problem, we will use the Bellman equation, which is the main concept behind reinforcement learning.

The Bellman Equation

The Bellman equation was introduced by the Mathematician Richard Ernest Bellman in the year 1953, and hence it is called as a Bellman equation.

It is associated with dynamic programming and used to calculate the values of a decision problem at a certain point by including the values of previous states.

It is a way of calculating the value functions in dynamic programming or environment that leads to modern reinforcement learning.

The key-elements used in Bellman equations are:

- Action performed by the agent is referred to as "a"
- State occurred by performing the action is "s."
- The reward/feedback obtained for each good and bad action is "R."
- A discount factor is Gamma "γ."

The Bellman equation can be written as:

$V(s) = \max [R(s,a) + \gamma V(s`)]$

Where,
V(s)= value calculated at a particular point.
R(s,a) = Reward at a particular state s by performing an action.
γ = Discount factor
V(s`) = The value at the previous state.

In the above equation, we are taking the max of the complete values because the agent tries to find the optimal solution always.

So now, using the Bellman equation, we will find value at each state of the given environment. We will start from the block, which is next to the target block.

For s3 block:

V(s3) = max [R(s,a) + γV(s`)], here V(s')= 0 because there is no further state to move.
V(s3)= max[R(s,a)]=> V(s3)= max[1]=> V(s3)= 1.

For s2 block:

V(s2) = max [R(s,a) + γV(s`)], here γ= 0.9(lets), V(s')= 1, and R(s, a)= 0, because there is no reward at this state.
V(s2)= max[0.9(1)]=> V(s)= max[0.9]=> V(s2) =0.9

For s1 block:

V(s1) = max [R(s,a) + γV(s`)], here γ= 0.9(lets), V(s')= 0.9, and R(s, a)= 0, because there is no reward at this state also.
V(s1)= max[0.9(0.9)]=> V(s3)= max[0.81]=> V(s1) =0.81

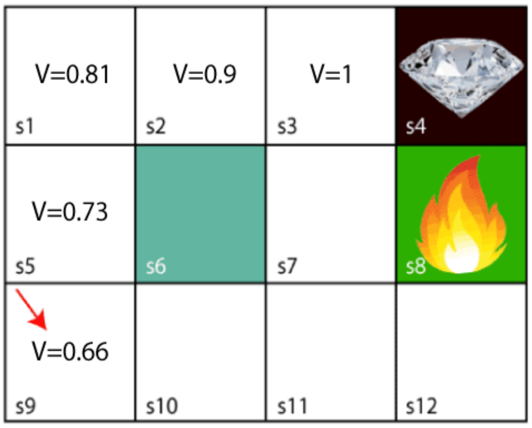
For s5 block:

V(s5) = max [R(s,a) + γV(s`)], here γ= 0.9(lets), V(s')= 0.81, and R(s, a)= 0, because there is no reward at this state also.
V(s5)= max[0.9(0.81)]=> V(s5)= max[0.81]=> V(s5) =0.73

For s9 block:

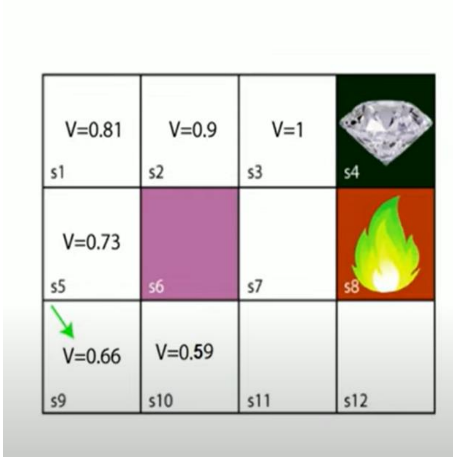
V(s9) = max [R(s,a) + γV(s`)], here γ= 0.9(lets), V(s')= 0.73, and R(s, a)= 0, because there is no reward at this state also.
V(s9)= max[0.9(0.73)]=> V(s4)= max[0.81]=> V(s4) =0.66

Consider the below image:



For S10 block:

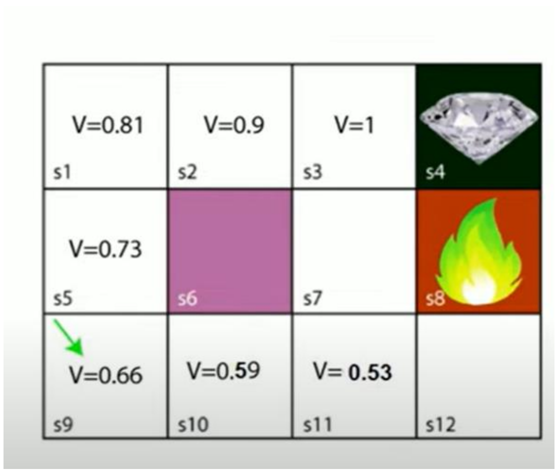
V(s10) = max [R(s,a) + γV(s`)],
here γ= 0.9(lets), V(s')= 0.66, and R(s, a)= 0, because there is no reward at this state also.
V(s10)= max[0.9(0.66)]=> V(s10)= max[0.59]=> V(s10) =0.59



For S11 block:

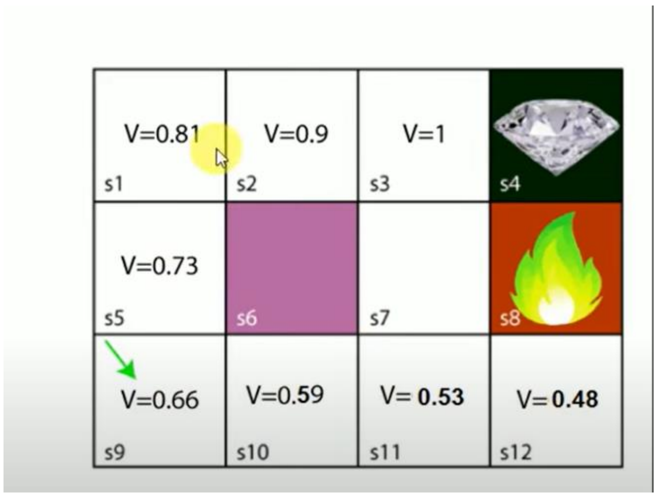
V(s11) = max [R(s,a) + γV(s`)], here γ= 0.9(lets), V(s')= 0.59, and R(s, a)= 0, because there is no reward at this state also.

- V(s11)= max[0.9(0.59)]=> V(s11)= max[0.53]=> V(s11) =0.53



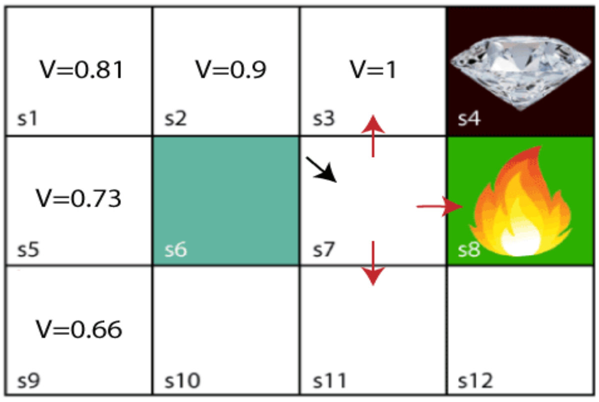
For S12 block:

V(s12) = max [R(s,a) + γV(s`)], here γ= 0.9(lets), V(s')= 0.59, and R(s, a)= 0, because there is no reward at this state also.
V(s12)= max[0.9(0.53)]=> V(s12)= max[0.48]=> V(s12) =0.48



So now, let's consider from the block next to the fire pit.

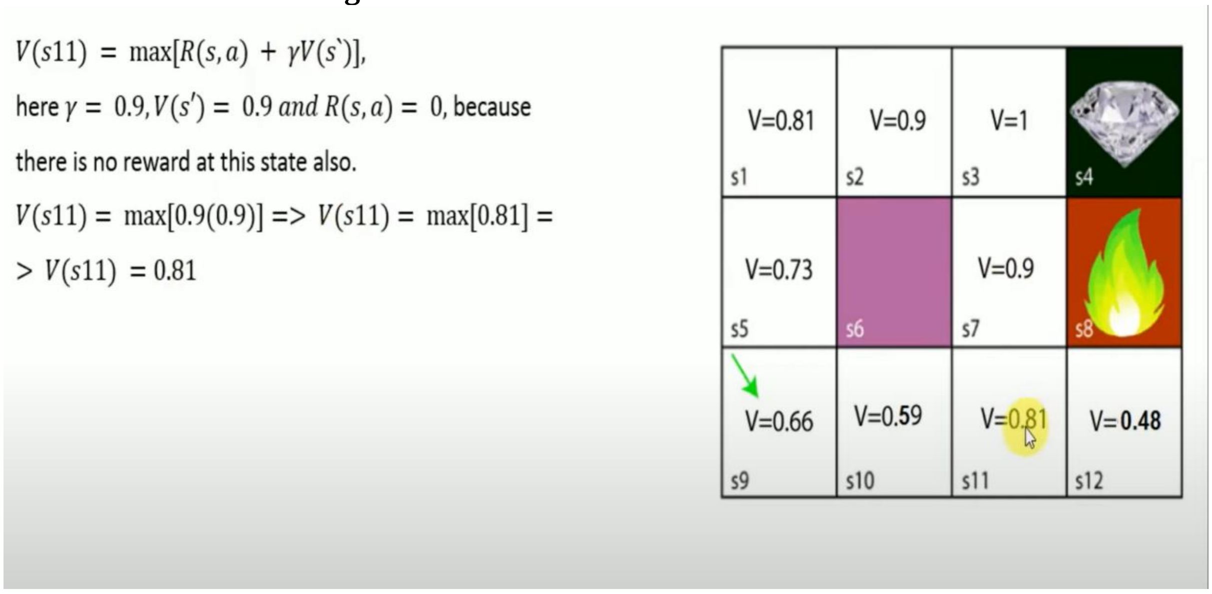
For S7 block:



Now, the agent has three options to move; if he moves to the blue box, then he will feel a bump if he moves to the fire pit, then he will get the -1 reward. But here we are taking only positive rewards, so for this, he will move to upwards only. The complete block values will be calculated using this formula.

- V(s7) = max [R(s,a) + γV(s`)],
- here γ= 0.9(lets), V(s')= 1, and R(s, a)= 0, because there is no reward at this state also.
- V(s7)= max[0.9(1)]=>V(s7)= max[0.9]=> V(s7) =0.9

For S11 block considering s7 as:



$V(s11) = \max[R(s,a) + \gamma V(s`)]$,

here $\gamma = 0.9, V(s') = 0.9$ *and* $R(s,a) = 0$, because

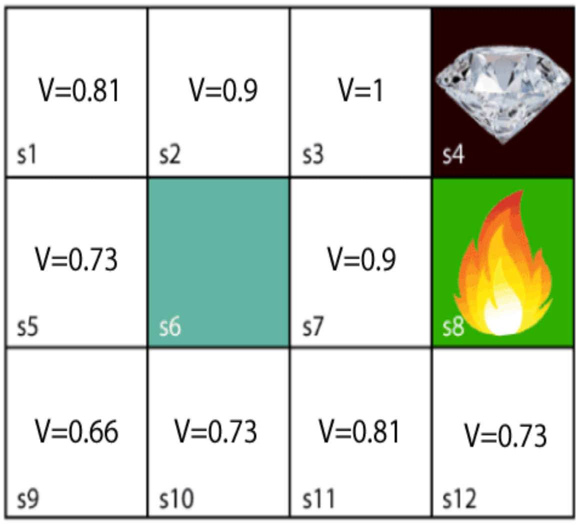
there is no reward at this state also.

$V(s11) = \max[0.9(0.9)] => V(s11) = \max[0.81] =$

$> V(s11) = 0.81$

- Repeat until, you get final max values for each state.
- After that when agent is in any state, it will reach to particular goal state by considering the optimal path.

Consider the below image:



How to represent the agent state?

We can represent the agent state using the Markov State that contains all the required information from the history. The State St is Markov state if it follows the given condition:

P[St+1 | St] = P[St +1 | S1,......, St]

The Markov state follows the Markov property, which says that the future is independent of the past and can only be defined with the present.
The RL works on fully observable environments, where the agent can observe the environment and act for the new state. The complete process is known as **Markov Decision process.**

Markov Decision Process

Markov Decision Process or MDP, is used to formalize the reinforcement learning problems. If the environment is completely observable, then its dynamic can be modeled as a Markov Process. In MDP, the agent constantly interacts with the environment and performs actions; at each action, the environment responds and generates a new state.

MDP is used to describe the environment for the RL, and almost all the RL problem can be formalized using MDP.

MDP contains a tuple of four elements (S, A, Pa, Ra):

- A set of finite States S
- A set of finite Actions A
- Rewards received after transitioning from state S to state S', due to action a.
- Probability Pa.

MDP uses Markov property, and to better understand the MDP, we need to learn about it.
Markov Property:

It says that *"If the agent is present in the current state S1, performs an action a1 and move to the state s2, then the state transition from s1 to s2 only depends on the current state and future action and states do not depend on past actions, rewards, or states."*

Or, in other words, as per Markov Property, the current state transition does not depend on any past action or state. Hence, MDP is an RL problem that satisfies the Markov property. Such as in a Chess game, the players only focus on the current state and do not need to remember past actions or states.

Finite MDP:

A finite MDP is when there are finite states, finite rewards, and finite actions. In RL, we consider only the finite MDP.

Markov Process:

Markov Process is a memoryless process with a sequence of random states S1, S2, ..., St that uses the Markov Property. Markov process is also known as Markov chain, which is a tuple (S, P) on state S and transition function P. These two components (S and P) can define the dynamics of the system.

Steps in Reinforcement Learning

- **Input:** The input should be an initial state from which the model will start
- **Output:** There are many possible output as there are variety of solution to a particular problem.
- **Training:** The training is based upon the input.
- The model will return a state and the user will decide to reward or punish the model based on its output.
- The model keeps continues to learn.
- The best solution is decided based on the **maximum reward**.

Passive and Active Reinforcement Learning

Both passive and active reinforcement learning are types of reinforcement learning.

In Passive reinforcement learning, the agent's policy $\pi(s)$ is fixed, which means that it is told what to do.

In Active reinforcement learning, the agent needs to decide or learn what to do as there's no fixed policy that it can act on.

The principal issue is exploration: an agent must experience as much as possible of its environment in order to learn how to behave in it. Hence, the goal of passive RL is to execute a fixed policy(sequence of actions) and evaluate it while that of an active RL agent is to act and learn an optimal policy. Ex: Chess game

Fully Observable environment(getting complete information through sensors).
In state s, agent already has a fixed policy $\prod(s)$ that determines its actions. Agent is therefore bound to do what the policy dictates, although the outcomes of agent actions are probabilistic. The agent may watch what is happening, so the agent knows what states the agent is reaching and what rewards the agent gets there. Goal: How good the policy is?
Each transition is annotated with both the action taken and the reward received at the next state. The agent used the information about rewards to learn the expected utility $U^{\prod}(s)$ associated with each nonterminal state, s.
The Utility is defined to be the expected sum of(discounted) rewards obtained if policy $^{\prod}$ is followed.

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1})\right]$$

Where $R(S_t,\prod(S_t),S_{t+1})$ is the reward received when action $\prod(S_t)$, is taken in state S_t and reaches state S_{t+1}. S_t is the state reached at time t when executing policy $\prod$.

Passive Reinforcement Learning Techniques

As the goal of the agent is to evaluate how good an optimal policy is, the agent needs to learn the expected utility Uπ(s) for each state, s. This can be done in three ways.

- Direct utility estimation
- Adaptive dynamic programming
- Temporal difference learning

Direct utility estimation:

In this method, the agent executes a sequence of trials or runs (sequences of states-actions transitions that continue until the agent reaches the terminal state). Each trial gives a sample value and the agent estimates the utility based on the sample's values. Can be calculated as running averages of sample values. The main drawback is that this method makes a wrong assumption that state utilities are independent while in reality they are Markovian. Also, it is slow to converge.

The utility of a state is defined as, the expected total reward from that state onward (called the **expected reward-to-go**). Estimate U∏(s) as average total reward of goals containing s (calculating from s to end of goal). At each trial the algorithm

- Calculates the observed reward-to-go for each state
- Updates the estimated utility for that state,

The computed expected reward can be applied directly to the observed data.

Adaptive Dynamic Programming (ADP):

ADP is a smarter method than Direct Utility Estimation as it runs trials to learn the model of the environment by estimating the utility of a state as a sum of reward for being in that state and the expected discounted reward of being in the next state.

It learns the transition model and solve the corresponding MDP and represent the transition model as a table of probabilities.

That is, the learned transition model P(s'|s, π(s)) and the observed rewards R(s) into the Bellman equation to calculate the utilities of the states.

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^{\pi}(s')$$

- Where,
- R(s) = reward for being in state s,
- P(s'|s, π(s)) = transition model,
- γ = discount factor and
- Uπ(s) = utility of being in state s'.

The environment is fully observable hence learning the model is easy. It can be solved using value-iteration algorithm. The algorithm converges fast but can become quite costly to compute for large state spaces. It adopts the approach of modified policy iteration. It updates the utility estimates after each change. ADP is a model-based approach and requires the transition model of the environment.

A model-free approach is Temporal Difference Learning.

Temporal difference Learning:

The Temporal difference (TD) is much simpler than direct utility estimation and adaptive dynamic programming. It is a model-free approach. It requires much less computation per observation. It does not need a transition model to perform its updates i.e., TD learning does not require the agent to learn the transition model. The update occurs between successive states and agent only updates states that are directly affected.

The environment itself supplies the connection between neighbouring states, in the form of observed transitions. It adjusts the values of the observed states.

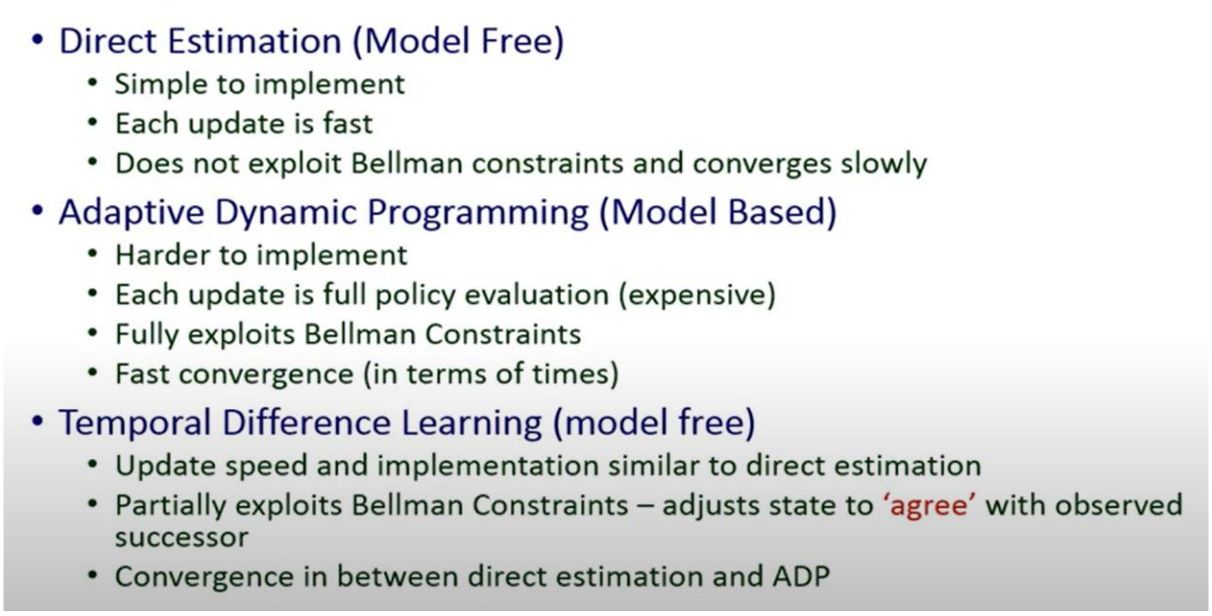
TD equation:

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha[R(s,\pi(s),s') + \gamma U^{\pi}(s') - U^{\pi}(s)]$$

Where α = learning rate

It uses the difference in utilities between successive states(s,s`). While ADP adjusts the utility of s with all its successor states, TD learning adjusts it with that of a single successor state, s'. TD is slower in convergence but much simpler in terms of computation.

Comparison



- Direct Estimation (Model Free)
  - Simple to implement
  - Each update is fast
  - Does not exploit Bellman constraints and converges slowly
- Adaptive Dynamic Programming (Model Based)
  - Harder to implement
  - Each update is full policy evaluation (expensive)
  - Fully exploits Bellman Constraints
  - Fast convergence (in terms of times)
- Temporal Difference Learning (model free)
  - Update speed and implementation similar to direct estimation
  - Partially exploits Bellman Constraints – adjusts state to 'agree' with observed successor
  - Convergence in between direct estimation and ADP

Active Reinforcement Learning Techniques

In this kind of RL agent, it assumes that the agent policy π(s) is not fixed. Agent is therefore not bound on existing policy and tries to act and find an Optimal policy for calculating and maximizing the overall reward value.

The agent learns the expected utility of each state and update its policy.

Techniques:

- **Q-Learning**
- **ADP with exploration function**

Q-Learning:

Q-learning is an Off-policy Reinforcement Learning algorithm, that seeks to find the best action to take given the current state which is used for the temporal difference Learning.

It is a **model free reinforcement leaning** where the agent only knows what are the set of possible states and actions and can observe the environment. So, the agent has to actively learn through the experience of interactions with the environment.

The agent will discover what are good and bad actions by trial and error. It learns the value function Q (S, a), which means how good to take action "a" at a particular state "s."

It is a TD learning method which does not require the agent to learn the transitional model, instead learns Q-value functions *Q(s, a)* .

$$U(s) = max_a\, Q(s,a)$$

Q-values can be updated using the following equation,

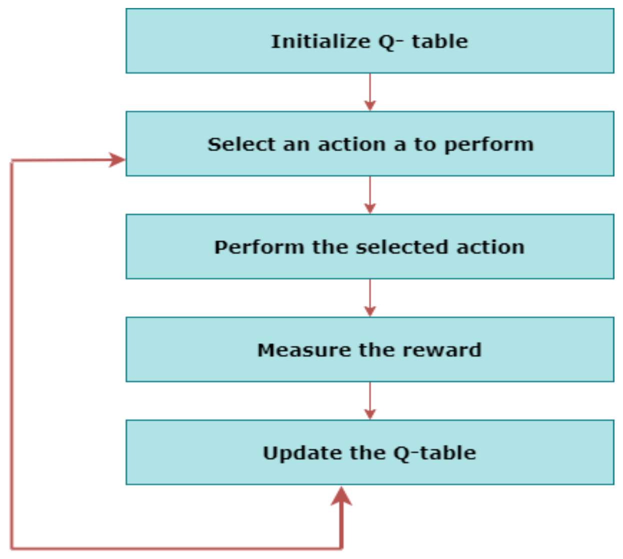
$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

Next action can be selected using the following policy,

$$a_{next} = arg\, max_{a'}\, f(Q(s',a'), N(s',a'))$$

It is considered off-policy because Q-Learning function learns from action that are outside the current policy like taking random actions and therefore a policy is not needed. It seeks to learn a policy that maximizes the total reward. It is a value-based RL algorithm which is used to find the optimal action selection policy using a Q function. **GOAL:** to maximize the value function Q.

The below flowchart explains the working of Q- learning:



When we start, all the values in the Q-table are zeros. There is an iterative process of updating the values. As we start to explore the environment, the Q-function gives us better and better approximations by continuously updating the Q-values in the table.

State Action Reward State action (SARSA):

- SARSA stands for State Action Reward State action, which is an on-policy temporal difference learning method. The on-policy control method selects the action for each state while learning using a specific policy.
- The goal of SARSA is to calculate the Q π (s, a) for the selected current policy π and all pairs of (s-a).
- The main difference between Q-learning and SARSA algorithms is that unlike Q-learning, the maximum reward for the next state is not required for updating the Q-value in the table.
- In SARSA, new action and reward are selected using the same policy, which has determined the original action.
- The SARSA is named because it uses the quintuple Q(s, a, r, s', a'). Where,
  s: original state
  a: Original action.
  r: reward observed while following the states
  s' and a': New state, action pair.

Now, we will expand the Q-learning.

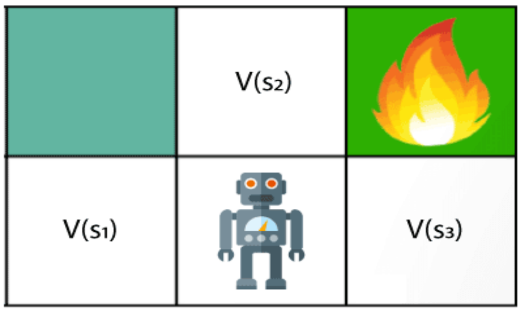
Q-Learning Explanation:

- Q-learning is a popular model-free reinforcement learning algorithm based on the Bellman equation.
- The main objective of Q-learning is to learn the policy which can inform the agent that what actions should be taken for maximizing the reward under what circumstances.
- It is an off-policy RL that attempts to find the best action to take at a current state.
- The goal of the agent in Q-learning is to maximize the value of Q.
- The value of Q-learning can be derived from the Bellman equation.

Consider the Bellman equation given below:

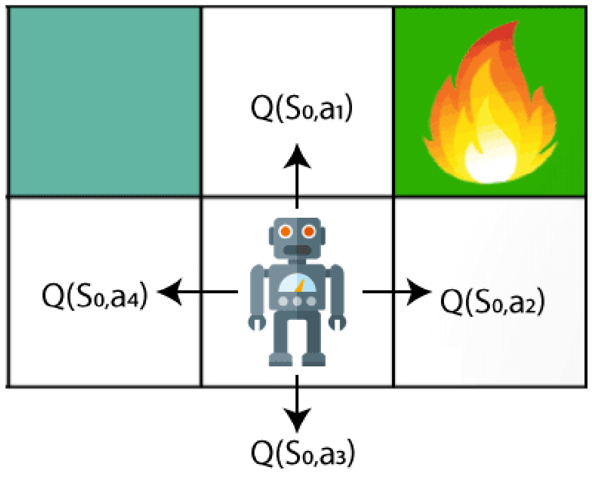
$$V(s) = \max [R(s,a) + \gamma \textstyle\sum_{s'} P(s, a, s')V(s`)]$$

In the equation, we have various components, including reward, discount factor (γ), probability, and end states s'. But there is no any Q-value is given so first consider the below image:



In the above image, we can see there is an agent who has three values options, V(s1), V(s2), V(s3). As this is MDP, so agent only cares for the current state and the future state. The agent can go to any direction (Up, Left, or Right), so he needs to decide where to go for the optimal path. Here agent will take a move as per probability bases and changes the state. But if we want some exact moves, so for this, we need to make some changes in terms of Q-value.

Consider the below image:



Q- represents the quality of the actions at each state. So instead of using a value at each state, we will use a pair of state and action, i.e., Q(s, a). Q-value specifies that which action is more lubricative than others, and according to the best Q-value, the agent takes his next move. The Bellman equation can be used for deriving the Q-value.

To perform any action, the agent will get a reward R(s, a), and also he will end up on a certain state, so the Q -value equation will be:

$$Q(S, a) = R(s, a) + \gamma \sum_{s'} P(s, a, s') V(s`)$$

Hence, we can say that, *V(s) = max [Q(s, a)]*

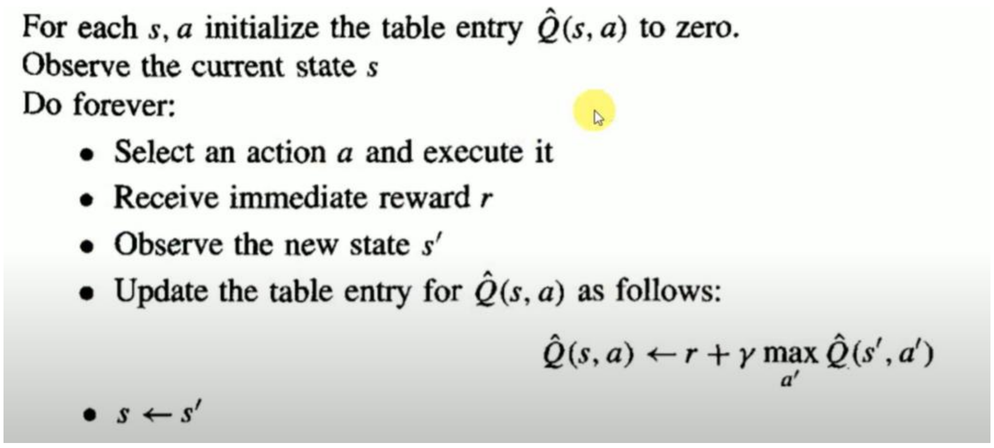
$$Q(S, a) = R(s, a) + \gamma \sum_{s'} (P(s, a, s') \max Q(s', a`))$$

The above formula is used to estimate the Q-values in Q-Learning.

What is 'Q' in Q-learning?

The Q stands for quality in Q-learning, which means it specifies the quality of an action taken by the agent. Quality represents how useful a given action is in gaining future rewards.

Q-Learning Algorithm



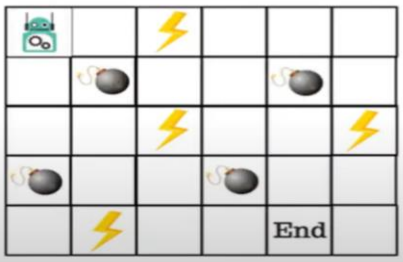
```
For each $s, a$ initialize the table entry $\hat{Q}(s, a)$ to zero.
Observe the current state $s$
Do forever:
    • Select an action $a$ and execute it
    • Receive immediate reward $r$
    • Observe the new state $s'$
    • Update the table entry for $\hat{Q}(s, a)$ as follows:
          $\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$
    • $s \leftarrow s'$
```

Q-Learning algorithm process

Step 1: initialize the Q-Table

We will first build a Q-table. There are n columns, where n= number of actions. There are m rows, where m= number of states. We will initialize the values at 0.

Actions :	↑	→	↓	←
Start	0	0	0	0
Nothing / Blank	0	0	0	0
Power	0	0	0	0
Mines	0	0	0	0
END	0	0	0	0



In our robot example, we have four actions (a=4) and five states (s=5). So we will build a table with four columns and five rows.

Q-table:

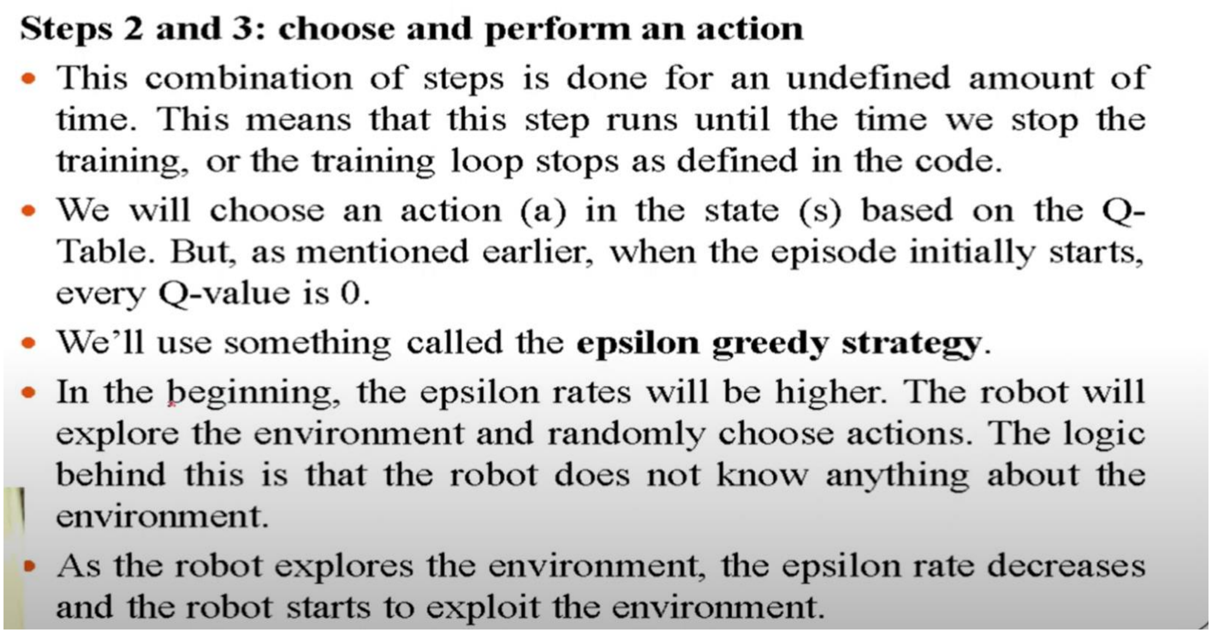
So, Q-table helps us to find the best action for each state. It helps us to maximize the expected reward by selecting the best of all possible actions. When Q-learning is performed, a Q-table or matrix is created that contains:

- No. of Rows = No. of states
- No. of Columns = No. of possible actions

Q-table is initialized with zero values. Q(state, action) returns the expected future reward of that action at that state. This function can be estimated using Q-Learning, which iteratively updates Q(s,a) using the Bellman equation.

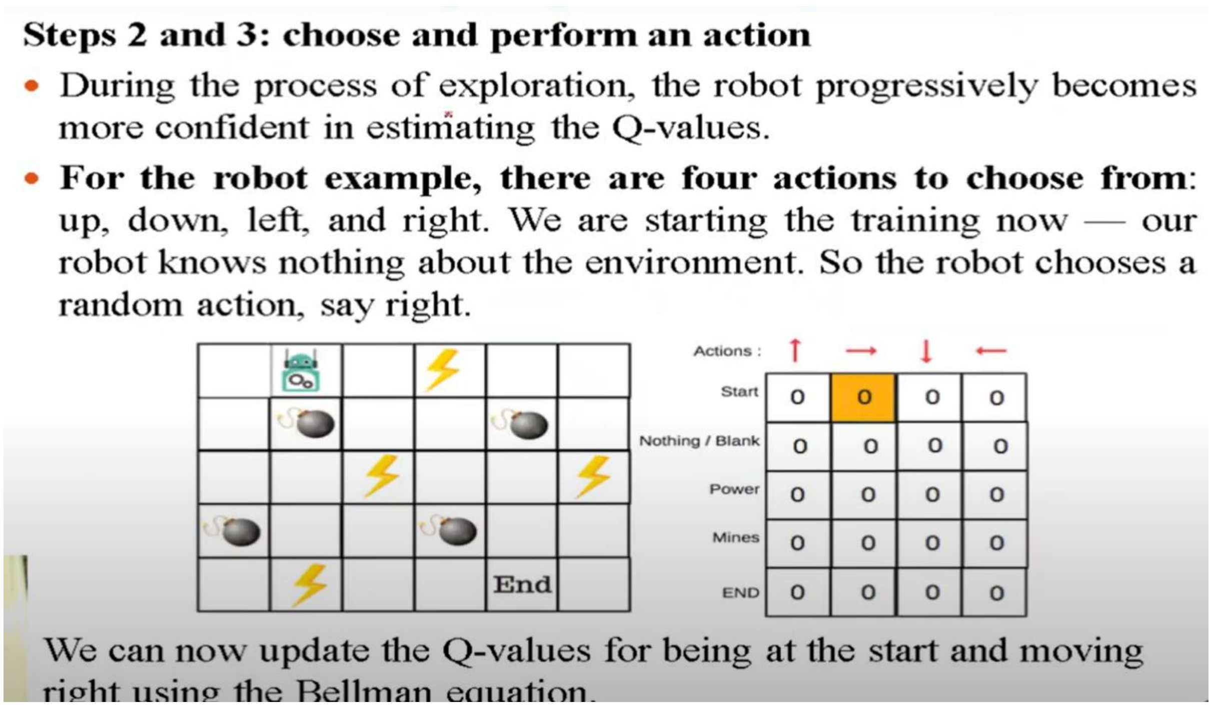
A Q-table or matrix is created while performing the Q-learning. The table follows the state and action pair, i.e., [s, a], and initializes the values to zero. After each action, the table is updated, and the q-values are stored within the table.

The RL agent uses this Q-table as a reference table to select the best action based on the q-values.



Steps 2 and 3: choose and perform an action

- This combination of steps is done for an undefined amount of time. This means that this step runs until the time we stop the training, or the training loop stops as defined in the code.
- We will choose an action (a) in the state (s) based on the Q-Table. But, as mentioned earlier, when the episode initially starts, every Q-value is 0.
- We'll use something called the **epsilon greedy strategy**.
- In the beginning, the epsilon rates will be higher. The robot will explore the environment and randomly choose actions. The logic behind this is that the robot does not know anything about the environment.
- As the robot explores the environment, the epsilon rate decreases and the robot starts to exploit the environment.

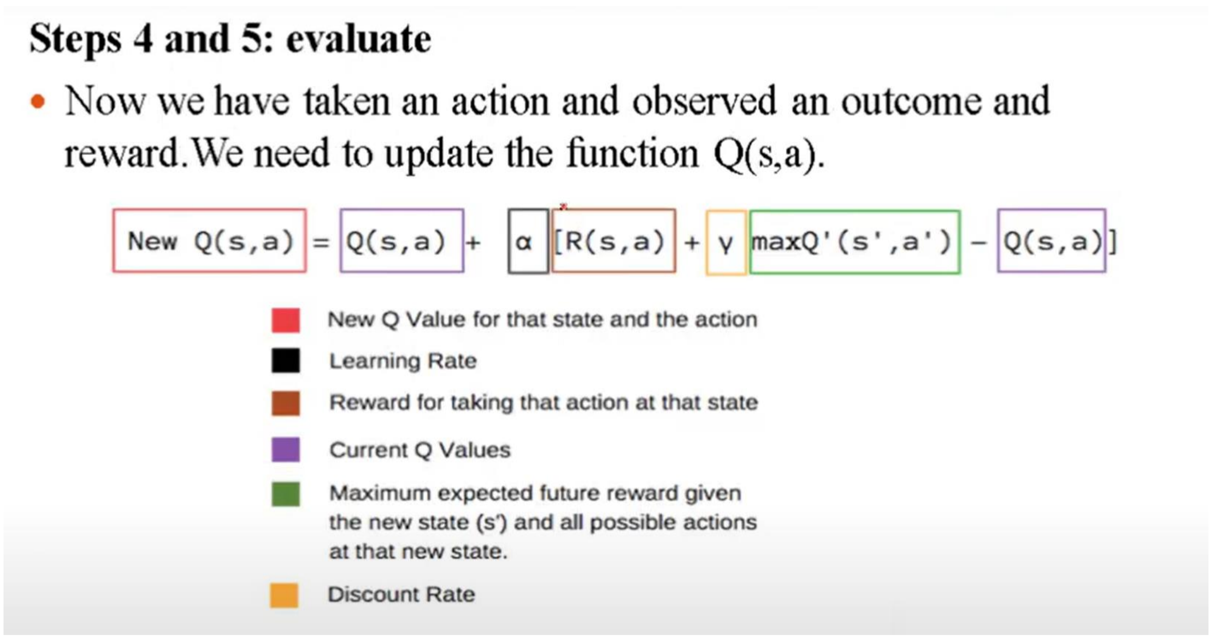


Steps 2 and 3: choose and perform an action

- During the process of exploration, the robot progressively becomes more confident in estimating the Q-values.
- **For the robot example, there are four actions to choose from**: up, down, left, and right. We are starting the training now — our robot knows nothing about the environment. So the robot chooses a random action, say right.

Actions :	↑	→	↓	←
Start	0	0	0	0
Nothing / Blank	0	0	0	0
Power	0	0	0	0
Mines	0	0	0	0
END	0	0	0	0

We can now update the Q-values for being at the start and moving right using the Bellman equation.

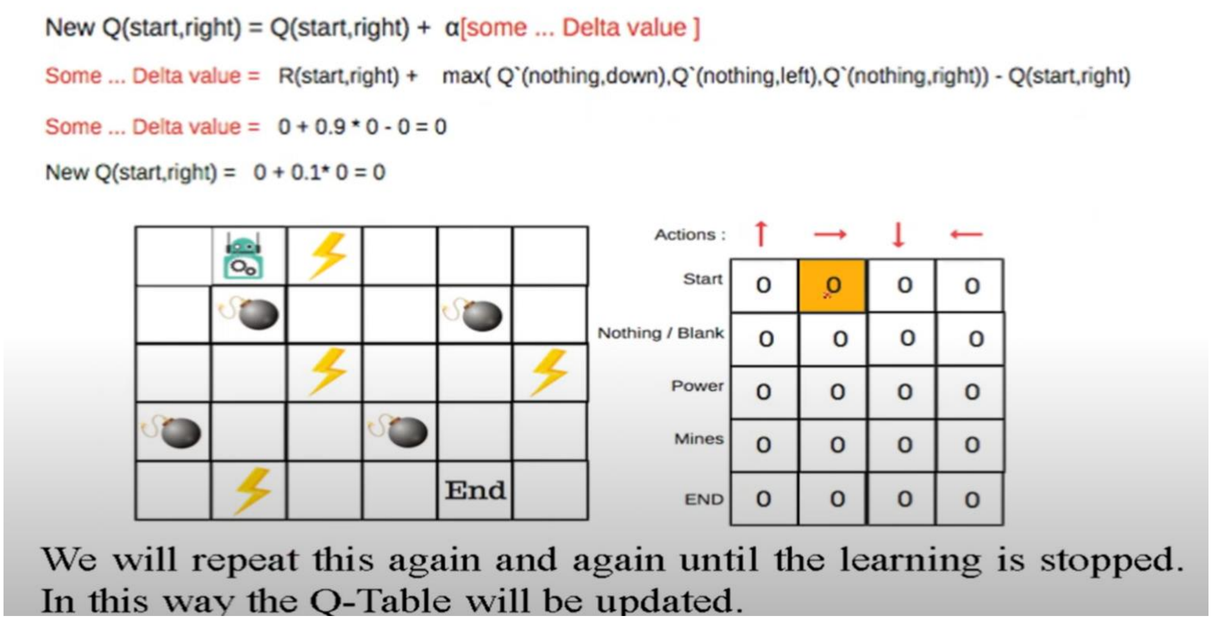


Steps 4 and 5: evaluate

- Now we have taken an action and observed an outcome and reward.We need to update the function Q(s,a).

$$\boxed{\text{New } Q(s,a)} = \boxed{Q(s,a)} + \boxed{\alpha}\,\boxed{[R(s,a)} + \boxed{\gamma}\,\boxed{\max Q'(s',a')} - \boxed{Q(s,a)}]$$

🟥	New Q Value for that state and the action
⬛	Learning Rate
🟫	Reward for taking that action at that state
🟪	Current Q Values
🟩	Maximum expected future reward given the new state (s') and all possible actions at that new state.
🟧	Discount Rate



New Q(start,right) = Q(start,right) + α[some ... Delta value]

Some ... Delta value = R(start,right) + max(Q`(nothing,down),Q`(nothing,left),Q`(nothing,right)) - Q(start,right)

Some ... Delta value = 0 + 0.9 * 0 - 0 = 0

New Q(start,right) = 0 + 0.1* 0 = 0

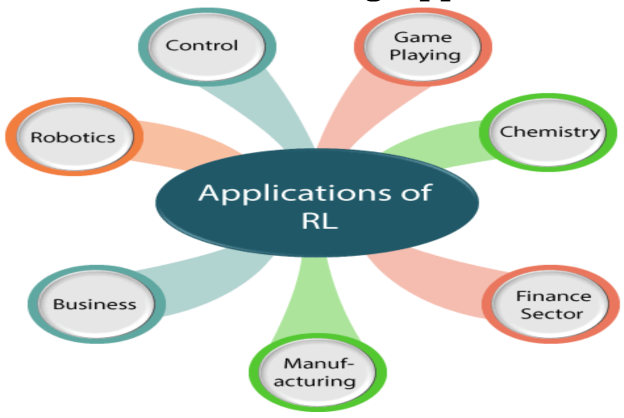
Actions :	↑	→	↓	←
Start	0	0	0	0
Nothing / Blank	0	0	0	0
Power	0	0	0	0
Mines	0	0	0	0
END	0	0	0	0

We will repeat this again and again until the learning is stopped. In this way the Q-Table will be updated.

Reinforcement Learning Algorithms

Reinforcement learning algorithms are mainly used in AI applications and gaming applications. The main used algorithms are:

Reinforcement Learning Applications



1. **Robotics:**
   RL is used in Robot navigation, Robo-soccer, walking, juggling, etc.
2. **Control:**
   RL can be used for adaptive control such as Factory processes, admission control in telecommunication, and Helicopter pilot is an example of reinforcement learning.
3. **Game Playing:**
   RL can be used in Game playing such as tic-tac-toe, chess, etc.
4. **Chemistry:**
   RL can be used for optimizing the chemical reactions.
5. **Business:**
   RL is now used for business strategy planning.

6. **Manufacturing:**
   In various automobile manufacturing companies, the robots use deep reinforcement learning to pick goods and put them in some containers.
7. **Finance Sector:**
   The RL is currently used in the finance sector for evaluating trading strategies.

ADP with exploration function

As the goal of an active agent is to learn an optimal policy, the agent needs to learn the expected utility of each state and update its policy.

Can be done using a passive ADP agent and then using value or policy iteration it can learn optimal actions. But this approach results into a greedy agent.

Hence, we use an approach that gives higher weights to unexplored actions and lower weights to actions with lower utilities.

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A} f\left(\sum_{s'} P(s'|s,a)U_i(s'), N(s,a) \right)$$

Where f(u,n) is the exploration function that increases with expected value u and decreases with number of tries n

$$f(u,n) = \begin{cases} R^+, if n < N_e \\ u, otherwise \end{cases}$$

R+ is an optimistic reward and Ne is the number of times we want an agent to be forced to pick an action in every state. The exploration function converts a passive agent into an active one.

Table: Comparison of active and passive learning methods

	Fixed Policy (Active)	**Policy not fixed (Passive)**
Model-free (real world)	Temporal Difference Learning (TD)	Q-learning
Model-based (simulation)	Adaptive Dynamic Programming(ADP)	ADP with proper exploration function